AD-A219 000

DTIC FILE COPY

(1)

SECURITY CLASSIFICATION OF THIS PAGE

DTIC ELECTE MAR 02 1990 S B D

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION: Unclassified			1b. RESTRICTIVE MARKINGS: N. A.	
2a. SECURITY CLASSIFICATION AUTHORITY: N A			3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N. A.				
4. PERFORMING ORGANIZATION REPORT NUMBER(S): Technical Report ONR-89-1			5. MONITORING ORGANIZATION REPORT NUMBER(S): Same	
6a. NAME OF PERFORMING ORGANIZATION: Netrologic, Inc.	6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research	
6c. ADDRESS (City, State, and ZIP Code): 5080 Shoreham Place, Suite 201, San Diego, CA 92122			7b. ADDRESS (City, State, and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217-5000	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research	8b. OFFICE SYMBOL (If applicable): Code 1142PS		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-89-C-0240	
8c. ADDRESS (City, State, and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217-5000			10 SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.: 61153N 42	PROJECT NO.: RR040209
			TASK NO.: RR04020901	WORK UNIT ACCESSION NO.: 400o042sbi01
11. TITLE (Include Security Classification): (U) Learning environment for neural networks and transient acoustics				
12. PERSONAL AUTHOR(S): Dan Greenwood, Fareed Stevenson, Rod Taber, Steve Deiss				
13a. TYPE OF REPORT: Final	13b. TIME COVERED: FROM 89/08/15 TO 90/02/14		14. DATE OF REPORT (Year, Month, Day): 90/02/14	15 PAGE COUNT: 36
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES: FIELD / GROUP / SUB-GROUP			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Sonar Signal Analysis, Intelligent Tutor, Neural Networks, Acoustic Signal Recognition, Underwater Signals	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

As a result of this Phase I research program, we were able to develop a unique workstation which will enable the user to easily interact with the relevant neural network technologies. The Tutor for Underwater Signature Analysis (TUSA) can predict a user's intentions and will provide assistance in an intelligent manner. Along with its intelligence, the system's other strength is its flexibility. TUSA can adapt or can be modified to suit the user's needs. In Phase I, we designed a framework which incorporates the fundamental elements of an intelligent tutor and we showed how these elements interact productively. Future work will complete TUSA's knowledge about backpropagation and underwater signals. *More important, however, will be the research into how multiple technologies can be controlled skillfully by a sonar operator to identify signals faster and more accurately. Combining conventional signal processing methods for detection and classification with advanced neural network paradigms will result in a very useful tool. Iterative improvement of TUSA's design through testing with sonar operators will provide a system which will also be applicable to many scientific and engineering fields.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL: John J. O'Hare	22b TELEPHONE (Include Area Code): (202) 696-4502	22c OFFICE SYMBOL: Code 1142PS

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

90 02 26 187

Table of Contents

1.0 INTRODUCTION TO THE TUTOR FOR UNDERWATER SIGNATURE ANALYSIS (TUSA) 1

2.0 BASIC CONCEPTS 1
2.1 Elements of Tutoring 2
2.1.1 Expert Module 2
2.1.2 Student Model 2
2.1.3 Tutoring Component 3
2.1.4 Communication Module 3
2.2 TUSA Design Limits 3
2.3 Critical Elements of TUSA 3
2.3.1 Meta-rules 3
2.3.2 Rules 4
2.3.2.1 Basic Backpropagation Network (BPN) Training Methods 4
2.3.2.2 Advanced BPN Training Methods 6
2.3.2.3 Special Variations of Backpropagation Relevant to Sonar Classification 10
2.3.3 The Concept of a Through-Focus Series 11

3.0 SOFTWARE IMPLEMENTATION 12
3.1 TUSA Workstation Prototype Platform 12
3.2 Flowchart of Control 12
3.3 Student-Computer Interaction Overview 14

4.0 EXPERIMENTS WITH A DATA SET 28
4.1 Signal Description 29
4.2 Signal Preprocessing 29
4.2.1 Training Results 30
4.3 Graphical Display of a Through-Focus Series 30
4.3.1 Accuracy 30
4.3.2 Noise Resilience 30

5.0 CAPABILITIES SUMMARY 33

6.0 RECOMMENDATIONS 34

7.0 REFERENCES 35

USE TITLE ON 1473 per Harold Hawkins
ONR/Code 1142PS
TELECON 3/1/90 CG

Table of Figures

Figure 1	TUSA Flow Chart	13
Figure W-1	TUSA Workstation Control Center Window Names	15
Figure W-2	Selecting Raw Data Source	16
Figure W-3	Selecting Raw Data Source	17
Figure W-4	System Deduces Necessary Network Architecture and Informs User	18
Figure W-5	Setting Network Architecture	19
Figure W-6	Setting Network Learning Rates	20
Figure W-7	Creating a Network to Specified Parameters	21
Figure W-8	Preparing Network for Training	22
Figure W-9	Creating Display Window for Network Training Output	23
Figure W-10	Creating Display Window for Network Training Output	24
Figure W-11	Creating Display Window for Network Training Output	25
Figure W-12	Creating Display Window for Network Training Output	26
Figure W-13	Workstation Displays After Network Has Finished Training	27
Figure 2	Taxonomy of Marine Signatures	28
Figure 3	Percent correct highest at (10-16) = 80%.	31
Figure 4	Percent correct highest at (20-2) = 81%.	32
Figure 5	With +/- 10% noise, percent correct highest at (20-2) = 80%.	32

1.0 INTRODUCTION TO THE TUTOR FOR UNDERWATER SIGNATURE ANALYSIS (TUSA)

The goal of NETROLOGIC's Phase I SBIR program for the Navy was to design an intelligent tutor to assist a sonar operator in the analysis of underwater signals generated by marine mammals and traffic. For this application, we defined the tutor to be an automated aid that either steers the student or provides computational support for a student while he is engaged in creating neural networks to recognize transient acoustic signals.

The current version of TUSA is a reactive learning environment able to perform elementary analysis of acoustic signatures and aid a student in developing the neural networks which analyze the underwater environment. TUSA is an intelligent help system that incorporates rules to aid in specifying a network.

2.0 BASIC CONCEPTS

In creating the TUSA software system, NETROLOGIC decided to place the emphasis on building a workable user interface that employs a relatively small number of network design rules to yield a system that is able to offer suggestions to a user.

To use TUSA, a student specifies an input file of formatted time domain signatures consisting of a variable number of class signatures and the size of a neural network of a particular type. The system then trains the requested network. When network training is complete, TUSA posits a series of other networks using the user specified network as a model. For example, if the user specified a network size of (20-3-5)[1], TUSA also trains a series of networks that range from 2-1-5 to 64-8-5[2]. It tests each network in the series for classification accuracy. It also attempts to determine the degree to which the network is memorizing the training samples, as opposed to generalizing. Output data is stored in a format which lends itself to rapid graphical display.

At this point, TUSA will have a better grasp of the classification problem for the samples than the student. An operative assumption of the tutor is that TUSA is always more informed than the student until the termination of the session, at which time the tutor summarizes the information so that, in effect, the student's knowledge equals that of the tutor for the data set.

TUSA then queries the student on the expected behavior of networks of differing sizes. Note that the backpropagation (BP) specification has 3 variables; the number of neurons in the input, hidden, and output layers. Each variable will be run through a do loop so that each network's performance can be easily tested by simply running the samples though the net and determining classification accuracy, resilience to noise, and the

---

[1]The user-specified configuration is termed the focal point. In Rumelhart notation for backpropagation, (20-3-5) ≡ 20 input units, 3 hidden units, 5 output units.

[2]The series is centered on (x-y-z) and ranging from (2-1-z) to (log xz - z) is called a through-focus series.

ability to generalize. Graphs of these parameters for a typical test case are given in Section 4.4.

2.1 Elements of Tutoring

Intelligent tutoring is still in its infancy, and at present there is no universally accepted definition of computer-aided tutoring. (Perhaps the best definition is given by Sleeman and can be found in [Kass][3].)

The oldest type of computer-aided instruction (CAI) is the generative model. The audience intended for this model are those students who must undergo repetitive conditioning such as grade school children learning multiplication. The next level is the adaptive CAI system that analyzes students' responses so it can tailor questions appropriately. Neither generative nor adaptive CAI have knowledge about the problem domain, which is ordinarily supplied by an outside source in a fixed format.

In [Self], tutoring involves three forms of knowledge:

- How to teach (including knowledge about students in general).
- What is being taught (domain knowledge).
- Who is being taught (student model).

The system that accomplishes these tasks contains an expert module, a student model, a tutoring component, and a communication module [Wolf].

2.1.1 Expert Module

This module contains what one would normally associate with a domain expert. In the case of underwater acoustic pattern recognition using neural networks, the module should cover both neural network knowledge and knowledge about underwater acoustics[4].

2.1.2 Student Model

A tutor should be able to tailor its question and answer sessions to students with varying degrees of expertise. There are several variations on how to accomplish this. The approach we favor at this time is to infer the expertise level by analyzing student responses to simple questions.

---

[3]This report will paraphrase and quote Kass's description of computer-aided instruction.

[4]TUSA currently does not use any knowledge about the signals themselves. It will work with any set of amplitude vs. time waveforms. However, ultimately (Phase II) some capability to suggest the kinds of noise present due to shallow focusing, convergence zones, layers of refractive index changes, and shipping noise, should be provided.

2.1.3 Tutoring Component

A good tutoring system, once it has estimated the expertise level of the student, should be able to decide what information to give the student, how it should be conveyed, and when it is appropriate. Equally important is the ability to let the student explore a little without affecting the tutor's estimation of a student's expertise level.

2.1.4 Communication Module

The desire for user-friendly communication is universal. However, attempts at natural language understanding can impose a severe penalty on a system. Translating English or a subset of standard English to a computer understandable language is very difficult and in many cases, not testable. Consequently, most initial tutors contain help screens and active manuals. This is clearly insufficient for unrestricted tutoring, and compromises must often be made.

There are numerous approaches to handling communication using windows and help screens. The passive approach is used to provide help only when requested by the student. The active approach is more effective since it monitors the interaction and makes suggestions when appropriate. Deciding when to communicate and how to communicate is frequently delegated to both the communication module and the tutoring component.

2.2 TUSA Design Limits

The design process culminating in an intelligent tutor can take a number of years depending on capabilities desired. In our application, the length and level of effort of Phase I precluded full scale development. To maximize our effectiveness, we followed several guidelines in our design and development of TUSA, namely:

- Select the most critical functions first.
- Favor functions that aid in building networks.
- Implement help screens.
- Code in a very high level language (SMALLTALK).

2.3 Critical Elements of TUSA

2.3.1 Meta-rules

Meta-rules in an AI system are rules that tell the system when to use ordinary rules. While this may be confusing, the following example will help clarify the concept.

> Rule 1: The network should be pruned to about half the number of hidden units and trained again to see the effect on performance.
>
> Meta-rule: Use Rule 1 when the network training method is BP and when training is finished.

Detailed factual knowledge is describable as ordinary rules, but there is a great deal of higher-level expertise which is of a procedural nature and is necessary in the correct

application of ordinary rules. Much of this knowledge is built in to our design. The user interface's organization, its patterns of interaction with the user, and the TUSA state are controlled by meta-knowledge coded into the software. As TUSA's flexibility increases, more meta-rules will be necessary to control the traffic of ordinary rules.

2.3.2 Rules

The current version of TUSA employs only a small number of the rules listed below. A much larger number of rules will be incorporated in Phase II.

Many rules were identified and, as development of the system advances during Phase II, rules appropriate to the system will be added. At this time, the rules we selected are not part of an expert system, but have been implemented within the code to which they are relevant. For example, when the user enters information which affects the architecture of the network, rules which are relevant to the event are queried and necessary feedback is provided to the user. In the future, if the complexity of the rule base requires it, they may be incorporated into a formal expert system.

Of all the contemporary neural network learning algorithms, backpropagation is the most widely studied and applied. Published literature includes reports of a great variety of attempts to speed up, analyze, configure and characterize this algorithm. In the following sections we will enumerate and describe the most relevant of these efforts. This information forms a data base from which we can formulate expert system rules to guide a neural network developer in the construction and training of a backpropagation network. For convenience we divide this review into three parts. First, we enumerate basic techniques widely used to configure and train networks. Second, we discuss more advanced techniques that are not widely accepted. Third, we describe the latest research on recurrent backpropagation in its algorithmic form and in its dynamical system form because these are very relevant to the sonar classification task. Familiarity with the basic backpropagation algorithm described in [Rumelhart] is assumed.

2.3.2.1 Basic Backpropagation Network (BPN) Training Methods

One basic technique sometimes overlooked by newcomers to the BPN paradigm is the use of batching [Rumelhart, McClelland]. This simply involves holding off all weight changes until the entire training file has been input, rather than updating the weights with the computed changes after each input is processed. This helps ensure gradient descent because without it, each weight adjustment would change the weight space over which we are trying to compute a gradient, and it would not be true gradient descent. It only makes sense when the training set is of a limited size and repeats with each new epoch.

Another of the most basic techniques used to speed convergence of BPNs is described in [Rumelhart, McClelland]. This method involves the use of a momentum term in the weight update equations. The momentum term represents the recent history weight changes so as to keep the gradient descent moving down the main gradient and not allow it to be pulled off course by minor one-shot anomalies in the weight adjustments. The form that this term takes depends on whether the training inputs are being batched or not. When batching, the weights are updated once every pass through the training file (epoch) by adding a portion of the last weight change and the new change (adjusted

by learning rate) to the old weight. When not batching, the weight update equation is adjusted to do the same thing after each input presentation. A simple variant of momentum reported in [Sejnowski] is to smooth the weight changes by partitioning each weight change into a portion based upon past history of changes, and the other portion on the current error. The only difference between this and the previous momentum method is that the scaling constants used to partition the weight update between the history and error must add up to one.

Symmetry breaking is used in standard BPN [Rumelhart, McClelland]. Here, the weight set is initialized to small random values so that one can be assured that the net input to the next layer will vary from unit to unit. If they all received the same input via identical weights or no input via zero weights, learning would not be possible.

Closely related to symmetry breaking is the technique of damaging the weights after training has reached a plateau. If the network is stuck in a local minimum, adding a small random amount to each weight might push it just far enough in weight space to allow it to recover and go on to a more global minimum. This technique is a feature of some BPN simulation environments such as SAIC ANSim [SAIC].

Another closely related technique involves mixing the input with a little bit of random noise before processing it [SAIC]. This forces the network to more fully explore the weight space as it proceeds through the training epochs. It thus increases the chances of finding a global minimum. The noise is usually provided according to a schedule of diminishing noise. After each epoch, the size of the range in noise values that will be added to each input is reduced. Intuitively, this is very similar to an annealing process.

Use of an offset for the sigmoid function to adjust its range between + and - .5 instead of 0 to 1 is another method that has become standard fare in BPN [Stornetta]. This also requires that inputs to the network and the target output be scaled between the same limits. Without input scaling, the inputs that are zero have no effect on the first layer weights since the weight changes are calculated as the outer product of the first hidden layer's delta terms with the input layer's activations. By adjusting everything to the +/- .5 range, all the binary or bipolar inputs will have an effect on first layer weights. This, in turn, tends to increase the rate of convergence of the network.

The offset method above naturally leads to the next technique, which is to adjust the training file by normalization so that all inputs and targets fall within the scaled range. The normalization multipliers used can be saved to allow the original data to be reconstructed, when necessary, from the normalized data. This input normalization method is used in commercial simulators such as [SAIC].

BPN convergence can be improved not only in terms of the number of iterations, but in terms of the real time taken for the simulation of it. This is done by only doing the backward error propagation when the error term is greater than some minimum, such as 0.1. In reality, this only saves time in certain implementations, which allow for the most general BPN architectures. These use a recursive algorithm for delta and error calculations where there is a choice of whether to recurse for a particular unit. For matrix multiply implementations of BPN, this speedup is not readily available unless all the error terms on a particular layer are less than the minimum.

The widely circulated BPN algorithm in [McClelland] includes calculation of gradient correlation to determine if the gradient is flattening out after each epoch. This is a useful tool for the experimenter and can be monitored automatically to warn of stagnation of the learning process that requires experimenter intervention.

It has been found by a number of research groups that it frequently pays to train a large network. After training, nodes and connections not instrumental to accuracy are eliminated. This has the effect of making the network generalize better. We have a metric for generalization which is simply the ratio

$$GM = (IS - PS) / IS$$

where IS is the initial (large) network size and PS is the pruned size. Refinement is on-going.

The maximum size of a BPN can be estimated by training a net to exactly reproduce the training set. In the neural network literature, this process goes by the name of the "encoder problem". The number of input and output nodes are made equal. The number of hidden nodes is modulated beginning with zero nodes up to a maximum. When the maximum is reached, the network is able to encode all information. Fewer nodes than this result in generalization (ordinarily a highly desired ability).

[Widrow] estimates the number of training samples (TS) needed by a simple formula given by $TS = NC/\epsilon$, where NC is the number of connections and ϵ is the required precision.

A final method in the basic set of BPN tools is to provide more than one way to detect when training has been successful. The usual method is to compute the RMS error over all the output units for a training epoch. An alternate method is to only require that the biggest error encountered for any output unit during an epoch be less than a criterion value. More sophisticated tests, tailored to specific application areas, are possible.

2.3.2.2 Advanced BPN Training Methods

The first method described here is to add noise to each weight. In [Von Lehman], it is reported that when noise is added to each weight (either digital or analog weights), learning rates much greater than unity can be tolerated without momentum and still achieve good results. The authors show that when the initial random weights and the weight noise meet certain restrictions related to the dynamic range in the weights, the probability of the network converging to a solution is greatly increased. Also, the dynamic range required of the weights can be relaxed when noise is present. The authors offer the intuitive explanation that using noise in weights having limited dynamic range forces the network to more fully explore the available solution space for a minimum.

It is obvious that the choice of initial network architecture can have a large effect on the ability of the network to converge on a solution based upon the number of hyperplanes (degrees of freedom) in the hidden units, and the number required by the training task itself. Therefore, there have been many suggestions on how to compute or evolve the correct network architecture for a given problem.

[Baum] offers a proof that one can have confidence approaching unity that a network will correctly classify future inputs if the network is trained to a calculated criterion level. The training set size for which this holds is a function of the weight set size, number of nodes, and the criterion level chosen. Therefore, given a choice of criterion level and number categories, it is possible to calculate the size of the network needed to generalize to the criterion level for those categories.

In [Sietsma], the authors show how a trained network can be manually pruned down to some minimal number of hidden units without adverse effects on performance. Training is done with a weight decay term that tends to make unused weights fall to zero Momentum and a term that adjusts the weight changes to be inversely proportional to the fan in of the unit that the weight links to are also used. The weights are then examined manually for unused weights. The least useful units indicated by these weights are removed, followed by further training to sharpen the accuracy. The technique does not specify what size network should be chosen to start from, except to note that there generally need to be more units to train than to actually do the recall portion of the task. It only illustrates how to prune the trained network.

In [Mozer], a "skeletonization" technique is illustrated which actually prunes a network automatically. This involves backpropagating an additional error term that expresses the performance of a network both with and without a particular unit. This is called the relevance measure. A relevance factor is kept for each unit, and as it drops toward zero, the unit becomes a candidate for removal. Removal of extra units in this way improves generalization and speeds up learning. The authors offer an intuitive interpretation of network behavior in terms of rules and exceptions. Removing irrelevant units has the effect of making the rules simpler yet more inclined to fail when an exception shows up.

Another technique [Chauvin] allows for the automatic identification of irrelevant hidden units. This involves computing a cost function that goes beyond squared error to include the energy of the hidden units. Energy is a function that represents the extent to which the unit activation is unchanging over the pattern training set. This is very similar to the relevance factor used in the skeletonization method above, and the expression used for backpropagation of errors is accordingly modified. This work does not go the additional step of automatic removal of units that are not contributing to the solution, but it achieves the same effect by reducing their weights to the point where they no longer play a part in network behavior.

In [Hansen], another objective function is proposed for BPN which enforces a weight decay bias for either individual weights or hidden units. When gradient descent is performed with this modified error function the number of hidden units in the network tends to decrease (as indicated by their outgoing weights approaching zero) while generalization improves.

In contrast to the above pruning methods, [Ash] defines an automatic method for adding more units to a network when learning hits a plateau. This method, called Dynamic Node Creation, makes it possible for the training process to recover from a poor initial network architecture that results in slow or no training. A new node is added to the hidden layer when the slope of the error curve flattens out. The method could possibly be extended to add additional layers, according to the author. Using the dynamic node

creation method seldom results in a network stuck in a local minimum over the cases tested by the authors.

The authors of [Kung] propose to use Algebraic Projection Analysis and show how it can be applied to find the optimal number of hidden units (for performance trials, not training) and to find the optimal learning rate. One result is that for irregular training sets of m inputs, m-1 is the optimal number of hidden units. Another is that an optimal learning rate can be computed which is proportional to the number of hidden units. The authors give results from empirical studies of networks of different sizes and using different learning rates that support these analytical results.

[Gutierrez] describes a heuristic procedure for calculating the number of hidden units needed in a network that was developed by experimentation. Calculations are all done based upon examination of the training file alone. The method tends to err on the high side by 10% and seldom underestimates in the cases tested. This is a good feature, since the extra hidden units may accelerate learning, and they can be pruned out with one of the other procedures that we have discussed.

Yet another method of improving generalization in BPNs is found in [Kruschke]. This involves the use of "bottlenecks" in the form of a hidden layer with a restrictive number of hidden units. As forward processing proceeds through the network and reaches this hidden layer, the network is forced to encode the input space in a restricted number of dimensions; ie., it is forced to generalize. This is similar to the effects observed in data compression networks trained by backpropagation [Cottrell]. However, the authors describe refinements to backpropagation which form the bottleneck without actually forcing activation to flow through a restrictive layer in terms of number of units. Rather, the networks form a functional bottleneck by adjusting the weights in the existing layers to achieve the same result. The benefit of this is that the bottleneck is represented in the weights in a distributed way. This makes the resulting network more immune to noise effects and resistant to real physical damage for the same reason that parallel distributed systems have those features in general.

A powerful training technique is described in [Waibel] for scaling up from small networks to larger ones with more functionality. For a speech recognition task, the author developed a network using network modularity and incremental design. This means training sub-networks on pieces of the problem, in this case recognition of small sets of phonemes, and then putting the smaller networks together and training on an enlarged training file. The weights in the pre-trained lower level networks are partially or totally frozen. The upscaled network adds new units with more weights, which are then tuned by training to integrate the responses from the lower level nets. Using this method, it was possible to achieve error rates much better than those obtained from other conventional methods. The intuitive appeal of this method can hardly go unnoticed.

[Cater] presents a method for adaptively adjusting the learning rate while training in order to achieve very high convergence rates for the network. The method requires keeping track of which training input produced the maximum output error during an epoch. During the next epoch, the training rate for that particular input is doubled. A second aspect of the adaptation is to decrease the learning rate used over the entire ensemble by 1/2 whenever the total ensemble error increases by 1%. This has the

effect of slowing the rate down to more carefully explore the local topology. Using these tricks, it is possible to increase the learning rates to 20 and more, in contrast to the small values $<< 1$ used by most other methods.

With the method of accelerated learning [Dahl], training time improvements have been reported in the range of an order of magnitude. Instead of evaluating the gradient of the error and then updating all the weights, this method reevaluates the error in successive points along the gradient to seek out the minimum quickly. After 4 to 8 such gradient steps, the procedure has usually hit a minimum and is ready for presentation of the next training input.

It has been argued that using 2nd derivatives with respect to the weights results in an optimal algorithm in the sense that the weights will move along the optimal path to the minimum [Parker]. This has been likened to minimization by Newton's method. However, the algorithm is computationally more expensive, and its value has to be measured on a case by case basis.

The gradient reuse algorithm [Hush] is similar in spirit to the method of accelerated learning above. Weight changes are repeatedly applied until they no longer produce a reduction in the output error. The size of the step along the gradient is adjusted dynamically so that it is largest on flat parts of the error surface and smallest when the gradient is steepest. It was reported that gradient reuse resulted in a 5-fold speedup of convergence for the traditional XOR benchmark. Complexity of the calculations for a particular training set depends upon the shape of the error surface in weight space because this determines the amount of gradient reuse that will be possible.

Extended backpropagation [Miikkulainen] involves backpropagating errors to a "concept" layer that precedes the input layer. The concept layer is a binary vector with a bit for every training category. When the weights between them and the input units are trained, the effect is the same as having the network learn its own representation of the outside world of inputs. In a concept learning example, a network was able to learn a distributed representation of several lexical concepts without being told what "features" it should focus on to do the partitioning. This resulted in a network with good damage resistance since the features that it was keying from in the input were not hard coded at the input layer. This approach fits very well with contemporary thought on the nature of language and semantics.

Our enumeration would not be complete without mention at least of the use of sigma-pi units and product units. Sigma-pi units were mentioned briefly in the original PDP treatise [Rumelhart]. However, few uses were given there. The idea was that the net input to a unit is the sum of several products, where the products represent the gating of one neural input by another. This is analogous to dendritic interactions known to exist, such as those of cortical pyrimidal cells. These units suffered the drawback that the number of weights to be adjusted grew too rapidly as the network size increased. Product units were recently introduced to overcome this failing while still providing units sensitive to other than simple linear combinations of the inputs [Durbin]. A product unit multiplies several inputs together, each weighted by a weight trained according to an alternate form of the traditional gradient descent algorithm. Alternating layers of standard and product units provide the same functionality as sigma-pi units. Examples

are given of cases where addition of these units results in a better solution for some known problems.

In summary, many methods have been reported for aid in the configuration and training of BPN networks. Some of the methods have potential for greatly speeding up the training process, while others offer the possibility of opening up new territory in what can be learned.

2.3.2.3 Special Variations of Backpropagation Relevant to Sonar Classification

Processing of temporal patterns, such as sonar data can be done with conventional backpropagation, given the proper preprocessing and representation of the input space using time warping, time delay networks and other techniques. However, there are several varieties of recurrent backpropagation being reported in literature, which present interesting capabilities for processing time varying signals.

In the first of these methods that will be described, conventional backpropagation is expressed in the form of dynamical system equations (coupled differential equations), rather than in the usual form of a recursive algorithm. Rather than "compute" a result, the dynamical system performs gradient descent by continuous time relaxation [Pineda]. Simulation of the relaxation of such a system is often more computationally complex than simple BPN. However, with the proper choice of physical implementation of the system using electronic components, very rapid convergence behavior should be possible. This first paper was primarily concerned with convergence to fixed points. However, it illustrates several features of the dynamical approach used by others for time varying signals.

In a second paper [Williams], an algorithmic technique is developed for recurrent backpropagation which does not incur the memory costs associated with the original recurrent method described in [Rumelhart], which used the technique of unwinding through time. The error term here is modified to reflect the difference between the actual output and the desired output at sampled points in time. Gradient descent is done for each time step along the trajectory. The final weight change is done based upon the accumulated changes along the trajectory. This method is still computationally costly because in a fully connected network with n units, the number of calculations grows with O(n**3).

In another dynamical translation of BPN [Pearlmutter] similar to [Pineda], coupled differential equations are used to perform gradient descent on the error surface for time varying target patterns rather than fortatic or terminal attractors. This system has been shown able to learn conventional problems like XOR, as well as cyclic behavior such as producing a circular or figure 8 trajectory through state space. Since the method involves numerical integration, it is presumed computationally costly, although convergence times measured in epochs were comparable to conventional backprop.

This, naturally, leads to the last special method to be included here. In [Owens], it is shown that conventional backpropagation is basically a fixed-step forward Euler integrator for a set of differential equations. Furthermore, these equations fall into the category of stiff differential equations, for which much better numerical solution methods exist. In particular, the authors describe much faster convergence using a

standard IMSL library stiff differential equation solver. Here, again, what is being learned is fixed points rather than trajectories, but the methods may be relevant for time varying input processing also.

These techniques, tricks, and new variations on BPN form a pool from which it is possible to draw techniques relevant to sonar classification for incorporation into the heuristic foundation of an intelligent tutor.

2.3.3 The Concept of a Through-Focus Series

A through-focus series is one of the most important concepts in TUSA[5]. It is the primary source of information for the tutor. With it, TUSA can coach the student through what we call a classification surface. For backpropagation, a surface might resemble Figure 3. The shape of the surface is the % Correct Classification as a function of both the number of hidden units and the number of input neurons. As one varies the two independent axes, the degree to which the network correctly classifies the signal set is plotted. Regions in which the signal set is exactly learned appear as level surfaces at or near the 100% Correct Classification mark. Peaks show regions of best classification, while valleys show the opposite. The goal of the user (although at the novice and intermediate levels he doesn't know it) is to find the point where classification is high and the cardinality of the hidden units and input units are simultaneously low.

A Classification Surface may be generated for noisy or clean signals. A surface from clean signals is distorted by noise. Rather than plotting a single 4-D surface, TUSA is able to plot a sequence of simulated 3-D surfaces. If it could plot simulated 4-D surfaces, the Classification Surface would be a function of input neurons, hidden neurons, and percent noise. As it is, TUSA generates a surface for each noise level[6] from 0% to 100%. The general effect of noise is to depress the surface leading eventually to a plane coincident with the plane determined by the horizontal axes. Somewhere during the addition of noise, a precipitous fall in classification is generally seen. This knee of the curve represents the entrance into a failure mode. Its position will be used by a Phase II prototype TUSA to assess the student's ability to build an "optimal" network for the signal set.

---

[5]In microscopy, a specimen is usually on the focal plane of the objective lens. In some cases, specimen geometry is ambiguous. If one knew exactly what the object looked like, it would be unnecessary to look at it again. Due to extensive specimen preparation, some parts of the specimen are in the focal plane and some are above or below it. Therefore, the microscopist photographs the image using several focus distances on both sides of true focus. The resulting photographs usually contain at least one acceptable copy. By analogy, we do not know the optimal size of the required network. We can experiment with sizes smaller and larger than the one requested by the user. Hence, we retain the terminology of the microscopist.

[6]Noise may be pseudo-white, pink, or multiplicative. At present, noise may be added in two ways: during signal construction via the analog mixer; or after corruption of the Discrete Fourier Transform power spectrum.

3.0 SOFTWARE IMPLEMENTATION

3.1 TUSA Workstation Prototype Platform

We used an 80386-based PC compatible for our system development. It operates at 20MHz and is supported by an 80387 math coprocessor, VGA graphics and a mouse. Four megabytes of memory are required to run our development environment: Smalltalk/V 286. This configuration provided a high-level development environment with good graphics and debugging features. We considered all popular procedure-oriented languages such as Fortran and "C" and then selected Smalltalk, primarily since it promised to minimize the time to achieve a useful prototype.

3.2 Flowchart of Control

The TUSA Workstation is decomposed into modules which are functionally independent. A clear definition of the behavior of each module and the interactions between modules is essential for the correct design and implementation of TUSA. Four modules are dedicated to neural network operations, one is dedicated to user inputs, one is dedicated to system outputs, and the last is an Expert module. TUSA's structure is shown in Figure 1. Descriptions of each of the data flow paths follow the figure.

Key Communication Event

1. User controls level of expert interaction/interference.
2. User input regarding network architecture, data manipulation, and system state control is sent to expert module for evaluation. Combined with the information about the existing network architecture (3), the current and previous system states (6), current data (4), current data manipulation method (5), and an expert-generated configuration, the expert module provides feedback to the user.
3. Current architecture is analyzed by the expert module.
4. Raw data are analyzed by the expert module.
5. Manipulated data and methods are analyzed by the expert module.
6. Current and historical states are analyzed by the expert module. Instruct user on device knowledge.
7. User controls display windows for network architecture, raw and manipulated data, and system state.

8. User specifies network architecture variables and controls (i.e. new).
9. User selects data source.
10. User specifies data manipulation methods (i.e. none, random order, bruise data function, manipulation function, reduce precision) and format (input and output width and height).
11. User specifies system state variable and controls (i.e. new, load, save, close, start, single data pass).
12. Architecture information is available for display.
13. Raw data is available for display.
14. Manipulated data is available for display.
15. State information is available for display.
16. Expert module outputs to a display window/pane.
17. Architecture information is made available when creating a system state via "new".
18. Raw data is used by data manipulation module.
19. Processed data is used by current state for training, verifying.

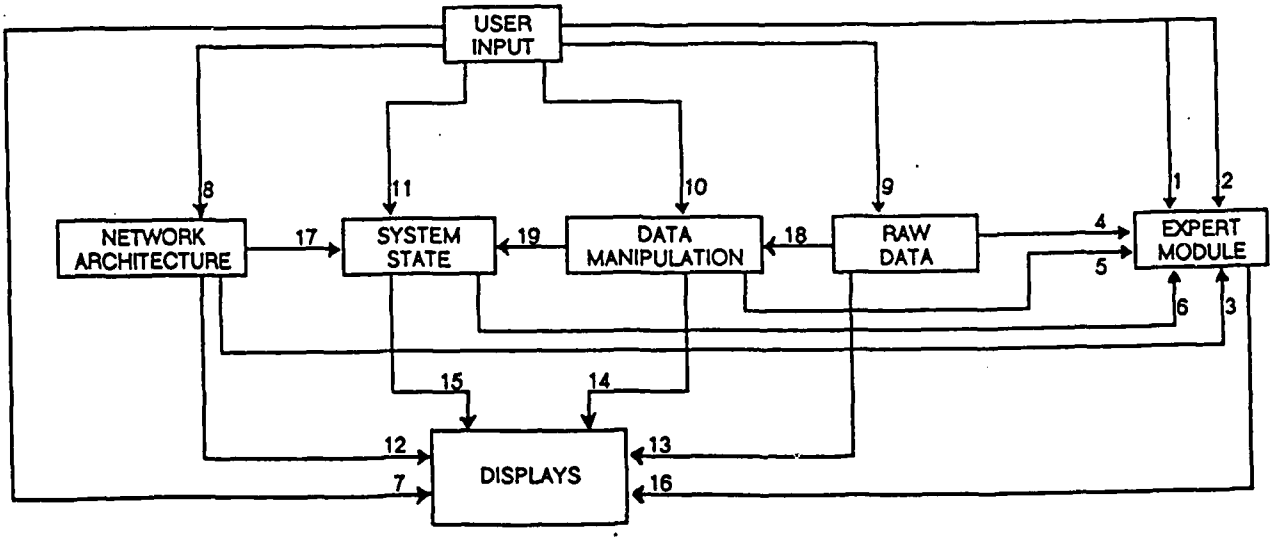

Figure 1 TUSA Flow Chart

3.3 Student-Computer Interaction Overview

A user interface has been designed which incorporates the best features from existing neural net environments while maintaining the simplicity of a menu-based system. The TUSA Workstation Control Center (Figure W-1) consists of four windows: two for user input and two for system output. The Main Menu Window lists the choices available to the user for control of five system modules. These five modules are: the Raw Data Module, the Data Manipulation Module, the System State Module, the Network Architecture Module, and the Displays Module. Two modules which are not accessible via the TUSA Control Center are the User Input Module and the Expert Module. The Branch Menu Window lists the choices available to the user after a selection has been made on the Main Menu. Choices in the Branch Menu may adapt as necessary to simplify operation of the Workstation. Adaptation reflects the user's history with the Workstation and current intentions. Workstation state information is displayed in the Reminder Window for the user's convenience. Upon user input, or under its own initiative, the system displays facts which pertain to the user's ongoing dialogue. The Response Window displays messages of a more permanent nature. The responses are generated by the same mechanism as the reminders, but the Response Window is a record of the dialogue.

Other types of input/output windows will be used as necessary. They display values as numbers of graphs, present lists of choices, or allow the user to inspect current parameters and set their value.

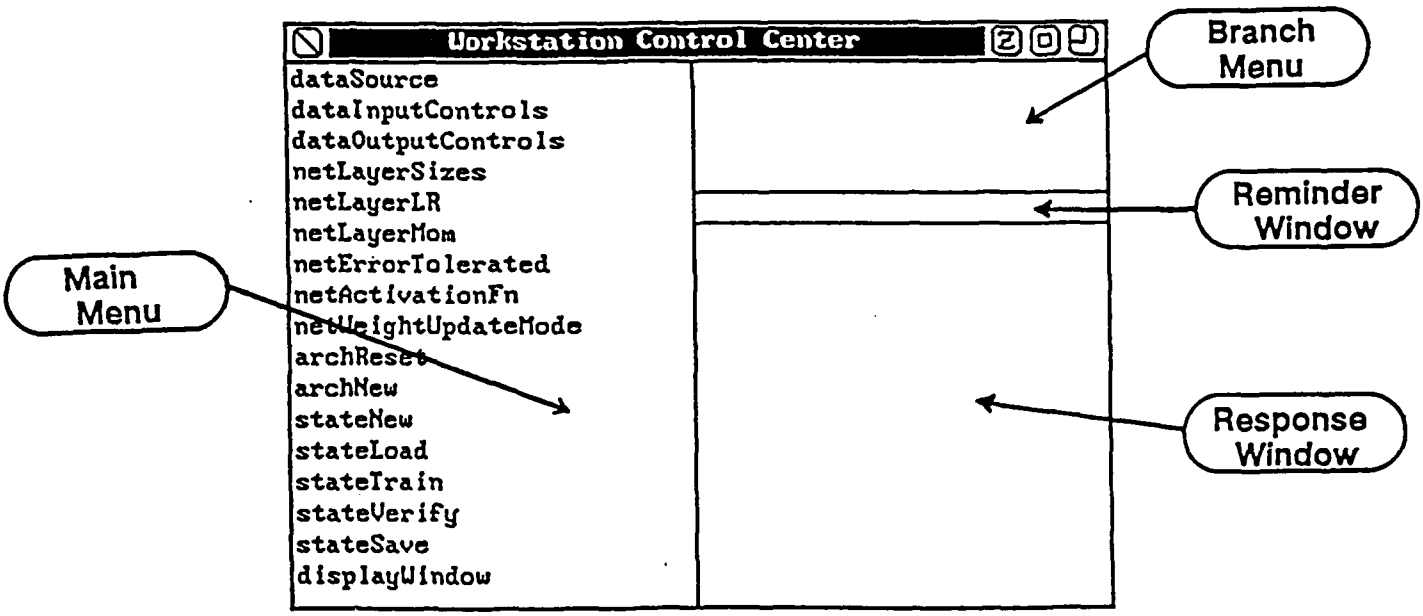

Figure W-1 TUSA Workstation Control Center Window Names

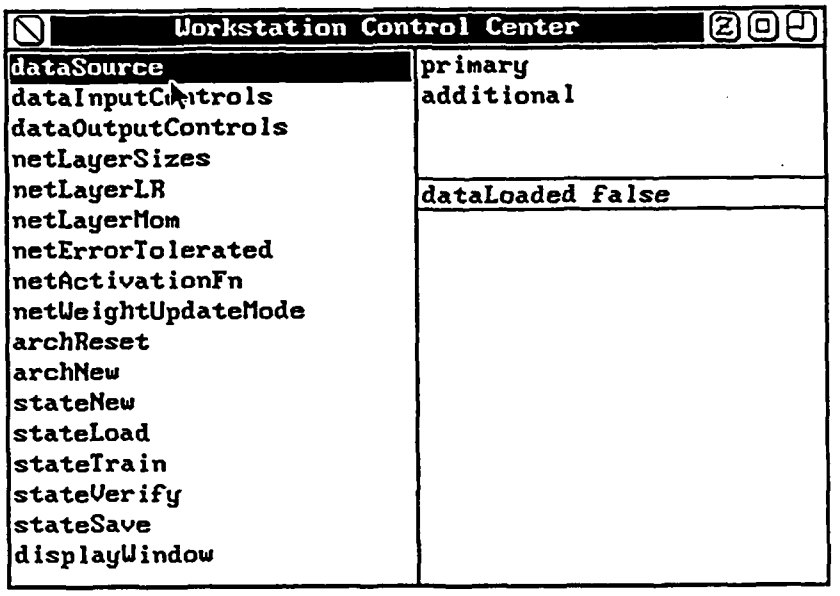

Figure W-2 Selecting Raw Data Source

USER INPUT	RESPONSE
User selects "data source" from main menu.	System branch menu is updated with "primary" and "secondary." The Workstation reminds the user that no data has been loaded and made available. The branch menu allows the user to specify which type of file is to be loaded. Primary files usually contain training data, whereas Secondary files usually contain verification data.

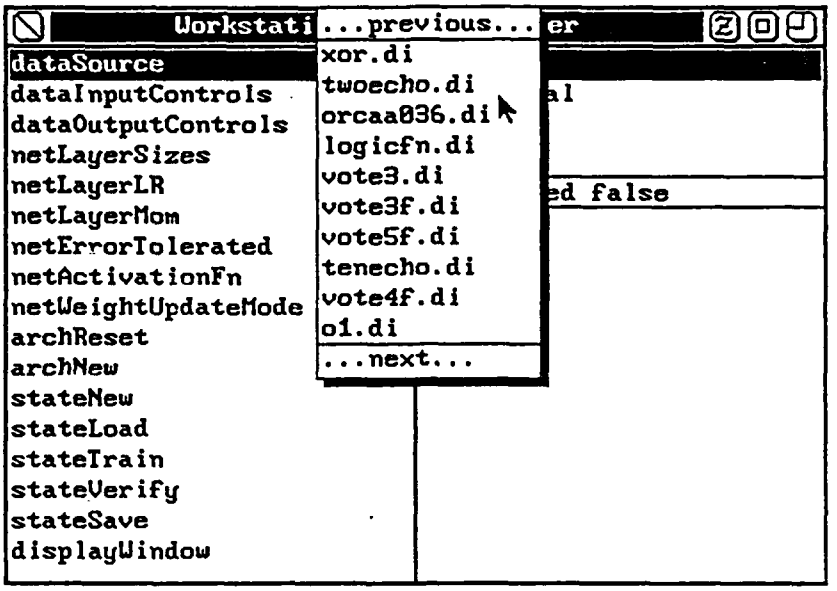

Figure W-3 Selecting Raw Data Source

USER INPUT	RESPONSE
User selects "primary" from system's branch menu.	A list selection window is created which contains Network Data filenames. User can select one or more of the filenames listed.
User selects "twoecho.di" from the list selection window.	

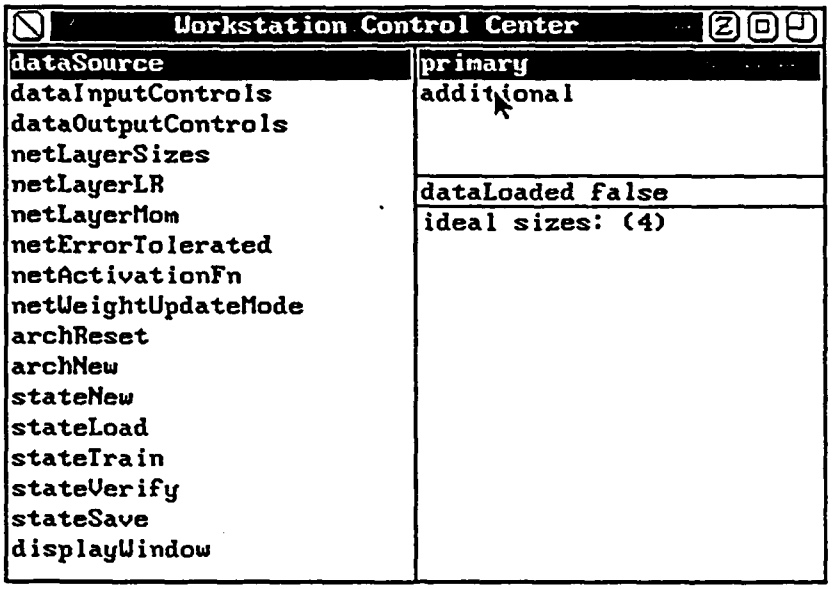

Figure W-4 System Deduces Necessary Network Architecture and Informs User

USER INPUT	RESPONSE
	The Workstation Expert Module infers that a backpropagation network of x-4-x architecture would provide the best performance.

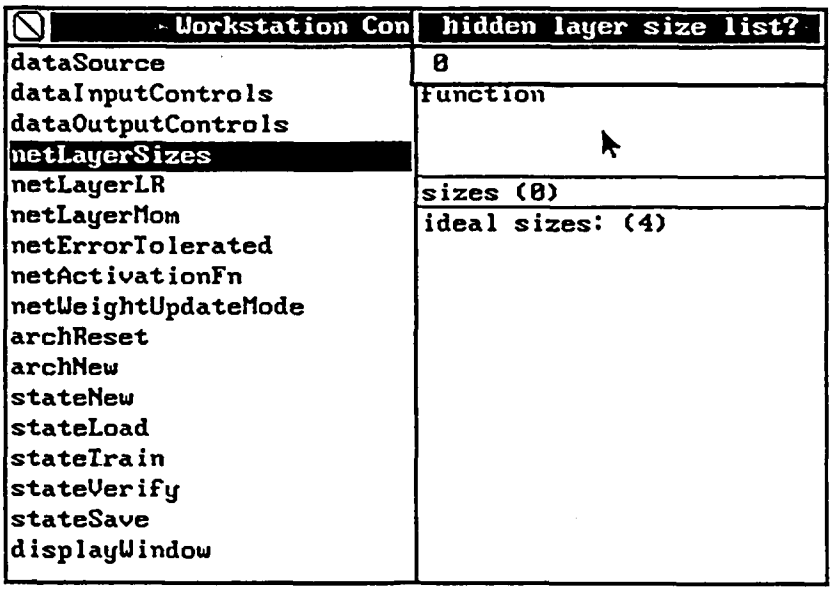

Figure W-5 Setting Network Architecture

USER INPUT	RESPONSE
User selects "netLayerSizes" from main menu.	The Workstation branch menu displays "constant" and "function."
User selects "constant" from branch menu.	System displays and makes available the number of hidden units for change in a Value Editing Window.

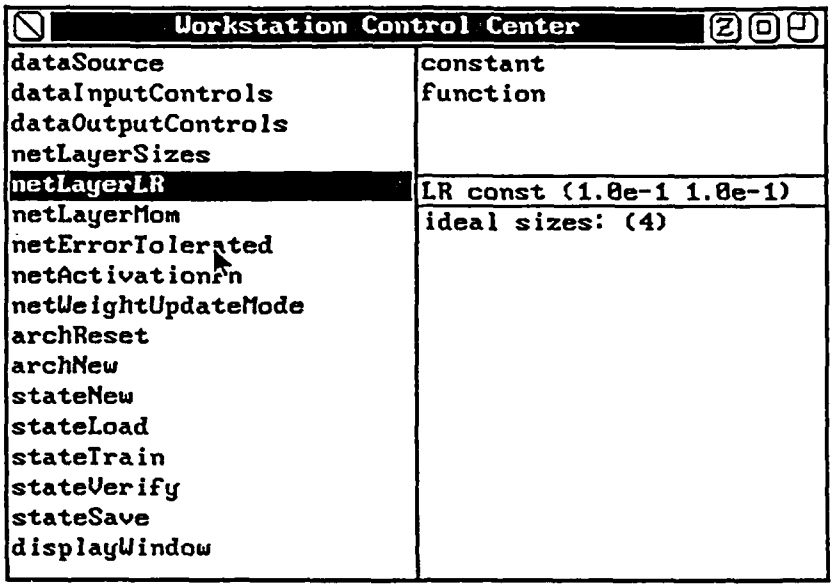

Figure W-6 Setting Network Learning Rates

USER INPUT	RESPONSE
User sets a value, "4," for the number of hidden units.	The system analyzes user input. In this instance, the value used was within the bounds expected, and no system response is created.
User selects "netLayerLR" from main menu.	The branch menu is set to "constant" and "function." The Reminder Window displays the BP layer learning rate. Since the default setting displayed in the Reminder Window is acceptable, user does nothing.
User selects "archNew" from main menu.	System updates branch menu to "OK."
User selects "OK" from branch menu.	System verifies all required architecture parameters are acceptable and then creates a network. The Response Window shows that the system accepted the architecture parameters.

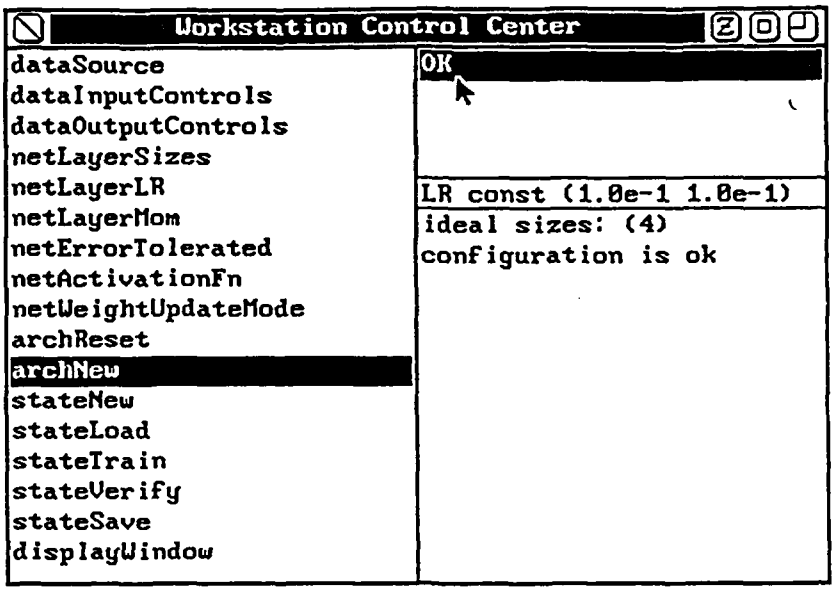

Figure W-7 Creating a Network to Specified Parameters

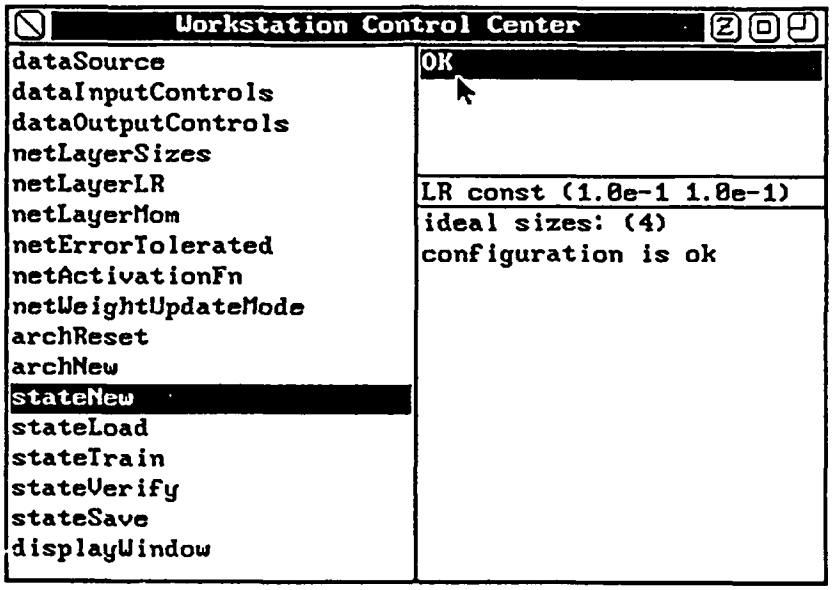

Figure W-8 Preparing Network for Training

USER INPUT	RESPONSE
User selects "stateNew" from main menu.	The system updates branch menu to "OK."
User selects "OK" from branch menu.	The system prepares for upcoming network training.

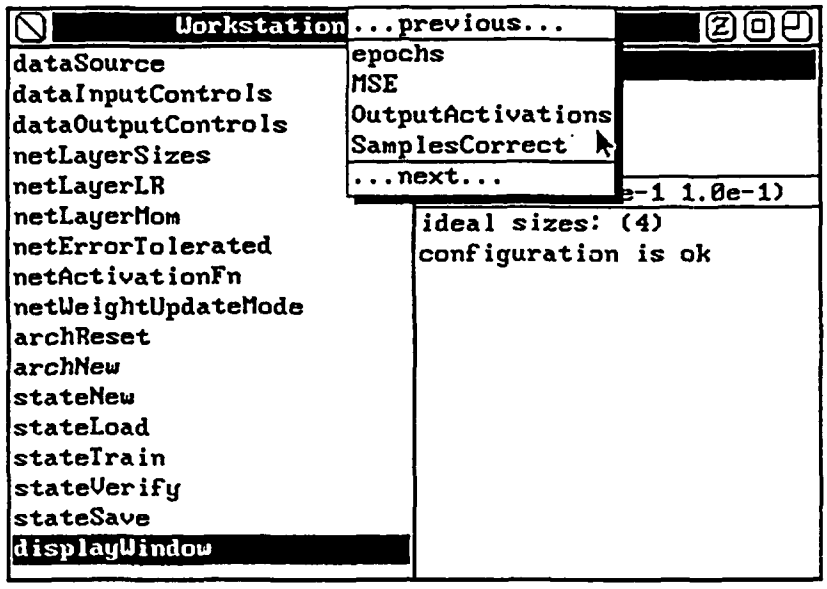

Figure W-9 Creating Display Window for Network Training Output

USER INPUT	RESPONSE
User selects "displayWindow" from main menu.	Branch menu displays "new," "existing," and "close."
User selects "new" from branch menu.	A list selection window appears listing the information available for display.

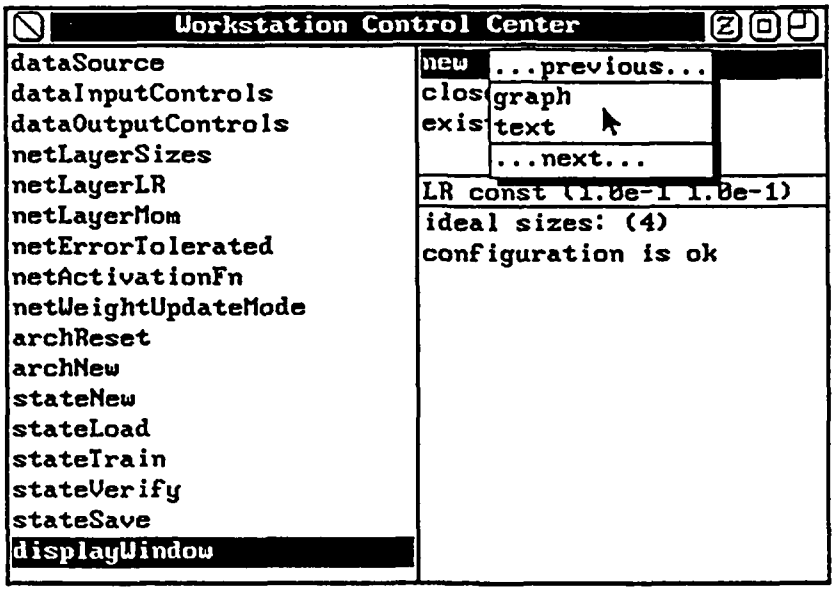

Figure W-10 Creating Display Window for Network Training Output

USER INPUT	RESPONSE
User selects "OutputActivations" from list selection window.	A list selection window appears listing the ways in which data can be displayed.

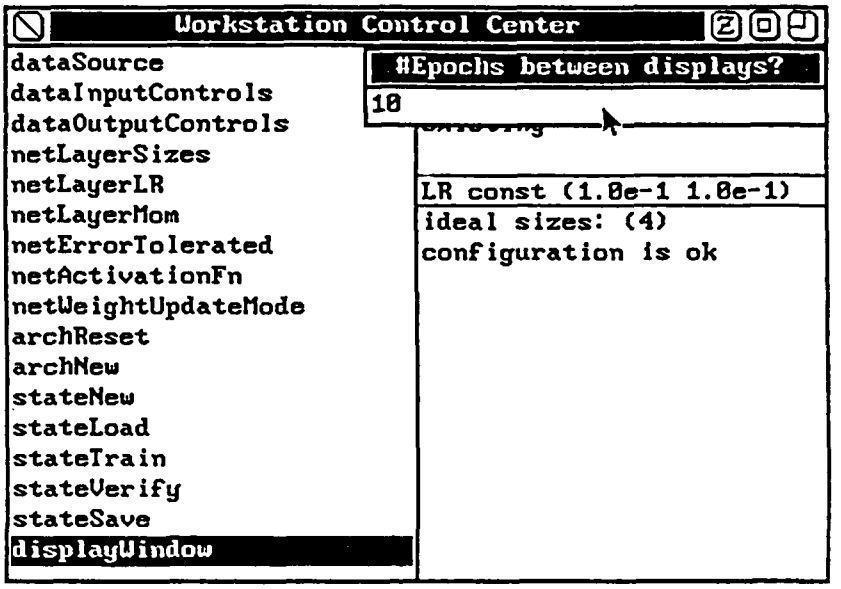

Figure W-11 Creating Display Window for Network Training Output

USER INPUT	RESPONSE
User selects "graph" from list selection window.	A value editing window requests the number of epochs between updates to the window being created.
User types in "10."	The user is prompted to select the position on the screen for the window being created.
User sets the window position.	The user is prompted to select the size for the window being created.

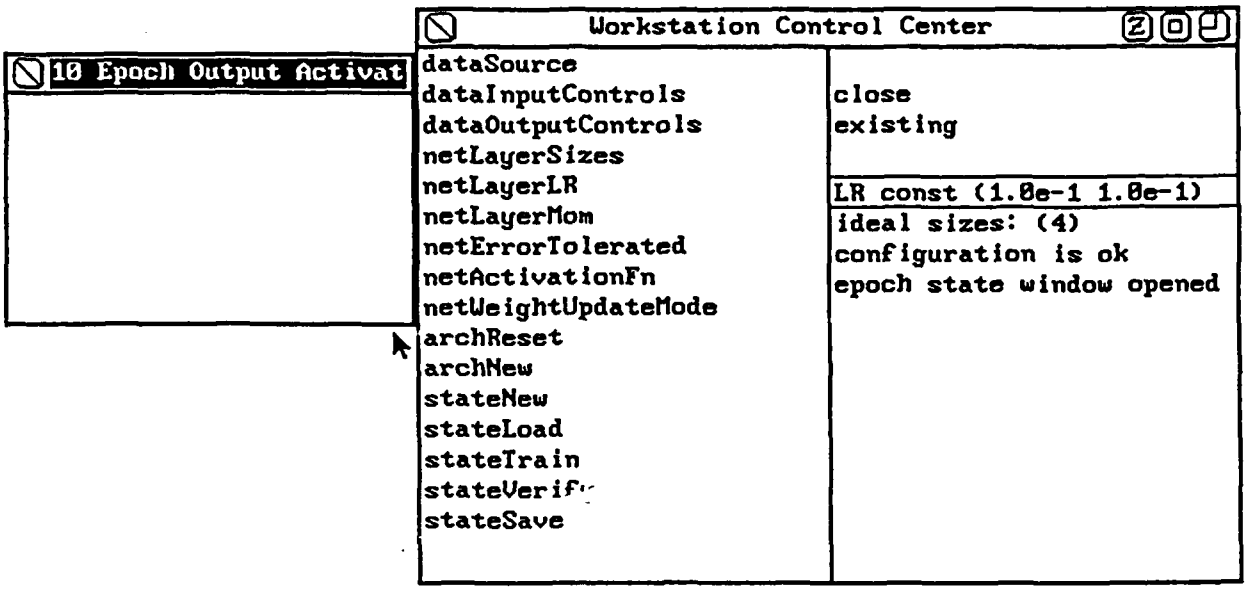

Figure W-12 Creating Display Window for Network Training Output

USER INPUT	RESPONSE
User sets the window size.	The system creates and labels the window. The Response Window records that the window was created.

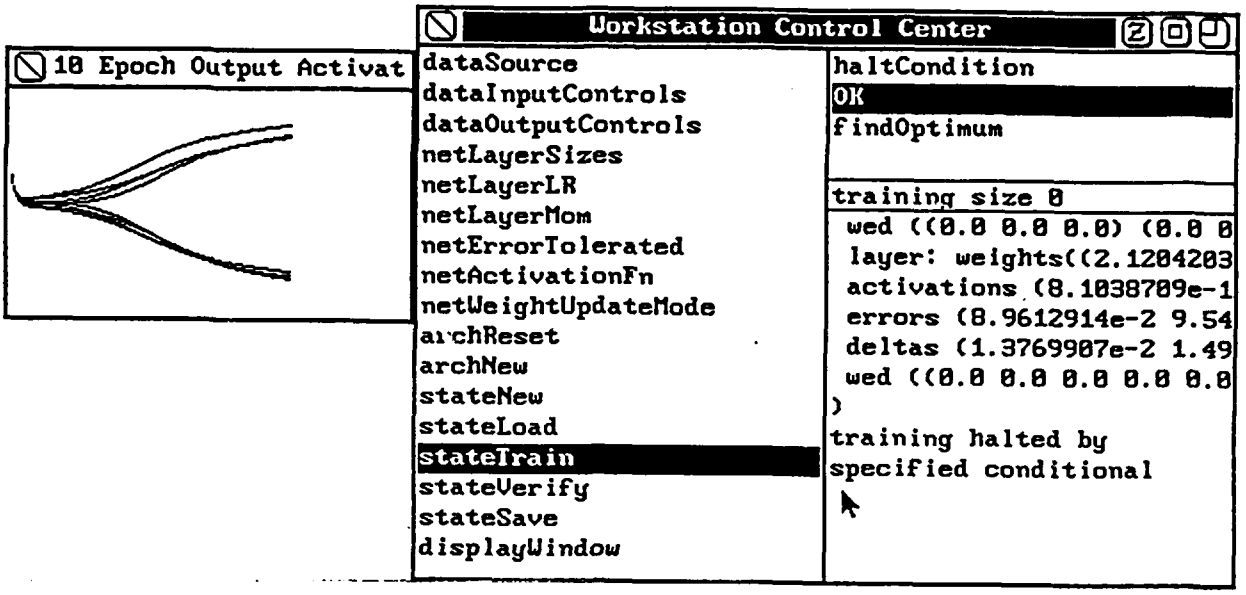

Figure W-13 Workstation Displays After Network Has Finished Training

USER INPUT	RESPONSE
User selects "stateTrain" from main menu.	Branch menu displays "haltCondition," "OK," and "findOptimum."
User selects "haltCondition" from branch menu.	A list selection window displays "epochErrorMax" and "length."
User selects "epochErrorMax" from list selection window.	A value editing window appears for the Maximum Output Unit Error to be acceptable.
User sets value to "0.1."	
User selects "OK" from branch menu.	The Workstation trains the network which has been previously readied and halts when no output units are above the threshold specified.

4.0 EXPERIMENTS WITH A DATA SET

Our sample problem domain of interest is underwater acoustics, specifically, marine mammal vocalizations. Figure 2 presents a taxonomy of marine signatures. Of interest is the lower box -- marine mammals.

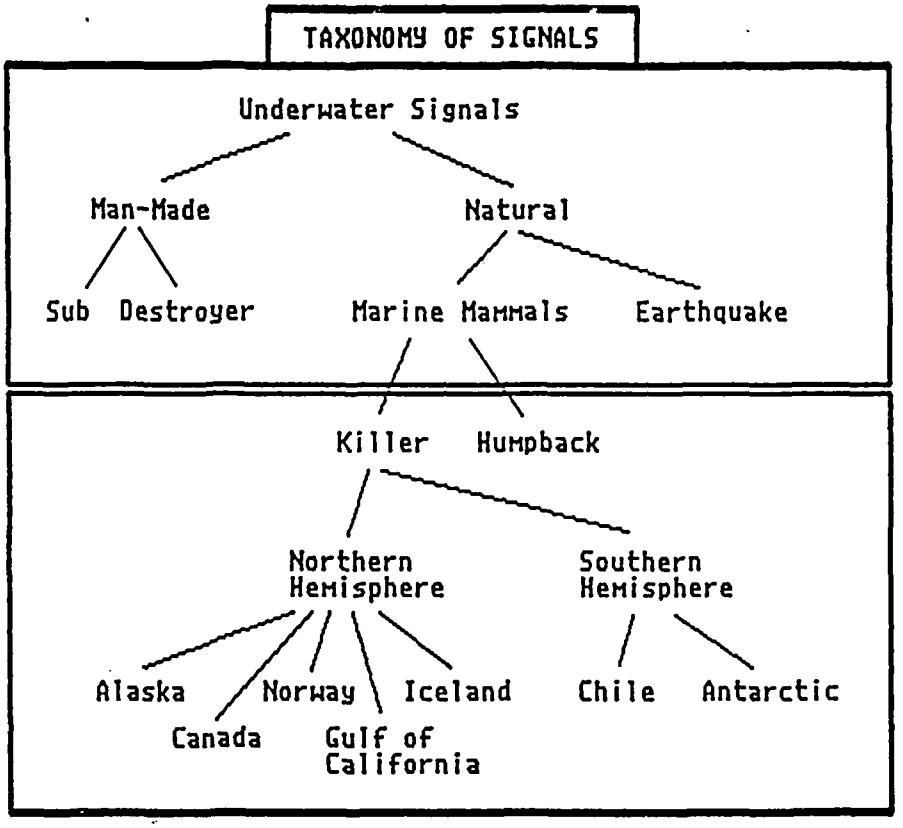

Figure 2 Taxonomy of Marine Signatures

At first glance, it should be relatively easy to separate the elements in the figure on the basis of acoustic content. However, the more one works with these signals, the more one realizes that the taxonomy confers a false sense of security. In reality, even clean signals often cannot be classified as belonging to one and only one of these categories, despite the fact that these are mutually exclusive categories. Why is this the case? On the surface, one expects sources with differing signal generators to produce distinguishing signatures.

That this is not the case is clearly demonstrated by anecdotal evidence given by sonar operators. For example, a Zodiac raft sounds very much like an aircraft carrier. The power plants and propulsion mechanisms are very different. Hinges on a dock[7] can

---

[7]A recording of this phenomenon by Dr. Rod Taber in San Diego Bay is available on audio cassette.

mimic marine mammals. Certain fish sound very mechanical[8]. Croakers sound like a very slow bilge pump.

One observation can be made. Natural sounds are usually frequency modulated, while artificial sounds are amplitude modulated. This is of help only with a clean signal and an index of modulation analyzer. The information is lost after a discrete Fourier transformation and other techniques have produced a digital record.

Neural networks promise new ways of recognizing such signals. We hope that models able to differentiate these signals will eventually be available.

4.1 Signal Description

From 1986 to Nov. 1988, a research team led by Dr. Rod Taber recorded approximately 100 hours of San Diego Bay noise. The Bay contains numerous boats, some captive marine mammals, and the usual complement of man-made and natural noise superimposed on the signatures. The location of the EDO hydrophone was on Shelter Island, opposite Lindbergh Field. Most recordings were made at the General Dynamics Sailing Club dock. The wooden dock platforms were held together by metal hinges that allowed the dock to move nondestructively with the tides.

These signals were used as noise overlays on killer whale recordings from the Hubbs Marine Research Institute, and three commercially available records[9]. We purchased a Realistic model 32-1100A stereo mixer. The whale call signal was attached to channel one and the San Diego Bay recordings attached to the other channel. The amount of graded noise added was controlled by the mixing control. Thus, it was possible to synthesize signatures of whales contaminated with varying degrees of ship traffic, shrimp, hinge, wind, and other noise. The sliding control on the 32-1100A allowed a continuum from zero noise to zero signal with peaks for either category.

The mixer output was recorded on a high fidelity VCR. The audio cassettes containing San Diego Bay noise were played on a Nakamichi CR-1A. The main amplifier was a JVC RX-777V with a graphic equalizer. Audio tape was metal with a frequency response of about 50 Hz to 17 KHz. The Sony recorder used at the site had a response of 10 Hz to 13 KHz.

4.2 Signal Preprocessing

We have converted audio samples of killer and humpback whales for network training. Sounds are converted at 20,000 samples per minute by an eight-channel, eight-bit A/D

---

[8]Listen to "Sounds of the Sea - Sonic Fish Sounds - Underwater Sounds of Biologic Origin," long playing record, Library of Congress Card number R-59-543, Folkways Records and Service Corporation. This recording was made by the Naval Research Laboratory.

[9]The NRL recordings mentioned previously, and the following:
- Playing Music with Animals , Jim Nollman, Folkways Records FX6118
- Songs of the Humpback Whale, Dr. Roger Payne, Capitol Records ST620

board and a ten-band equalizer. Converted values range from -127 to +128. Considering the frequency response characteristics of the equipment, we decided to use a cutoff of 8 KHz. This would reduce or eliminate the high frequency roll-off effect[10] of the recorders and players.

4.2.1 Training Results

To show that the data we collected was useful, a backpropagation network was trained to recognize humpback and killer whale signatures. Thirty samples of each type were used. The BP network architecture was 6667-10-4: a fully connected, feed-forward net with learning rates = 0.1 and no momentum. One-third of a second samples of raw data were used. The network correctly classified all whale calls it was trained on. It does not appear to generalize to novel data, but additional preprocessing should relieve this problem.

4.3 Graphical Display of a Through-Focus Series

The plots below show how classification performance is affected by the network architecture. On the horizontal axis, we show the varying number of input units. On the vertical axis, we show the varying number of hidden units. The height of the point plotted indicates the percent of whale signal which were correctly classified. Peaks show the combination of input units and hidden units that provide the best performance. The graphs are generated after only partial training, it would take too long and make any further activity pointless if we trained all combinations to their maximum performance. Although the data is, therefore, only an approximation, it identifies the most promising configurations for further investigation.

4.3.1 Accuracy

See Figures 3 and 4 for surface plots of Percent Correct Classification.

4.3.2 Noise Resilience

The plot in Figure 4 is related to that in Figure 5. The latter shows the same trained networks with their classification performance to signal which have been modified with +/- 10 percent noise.

---

[10]That is, no compensation necessary.

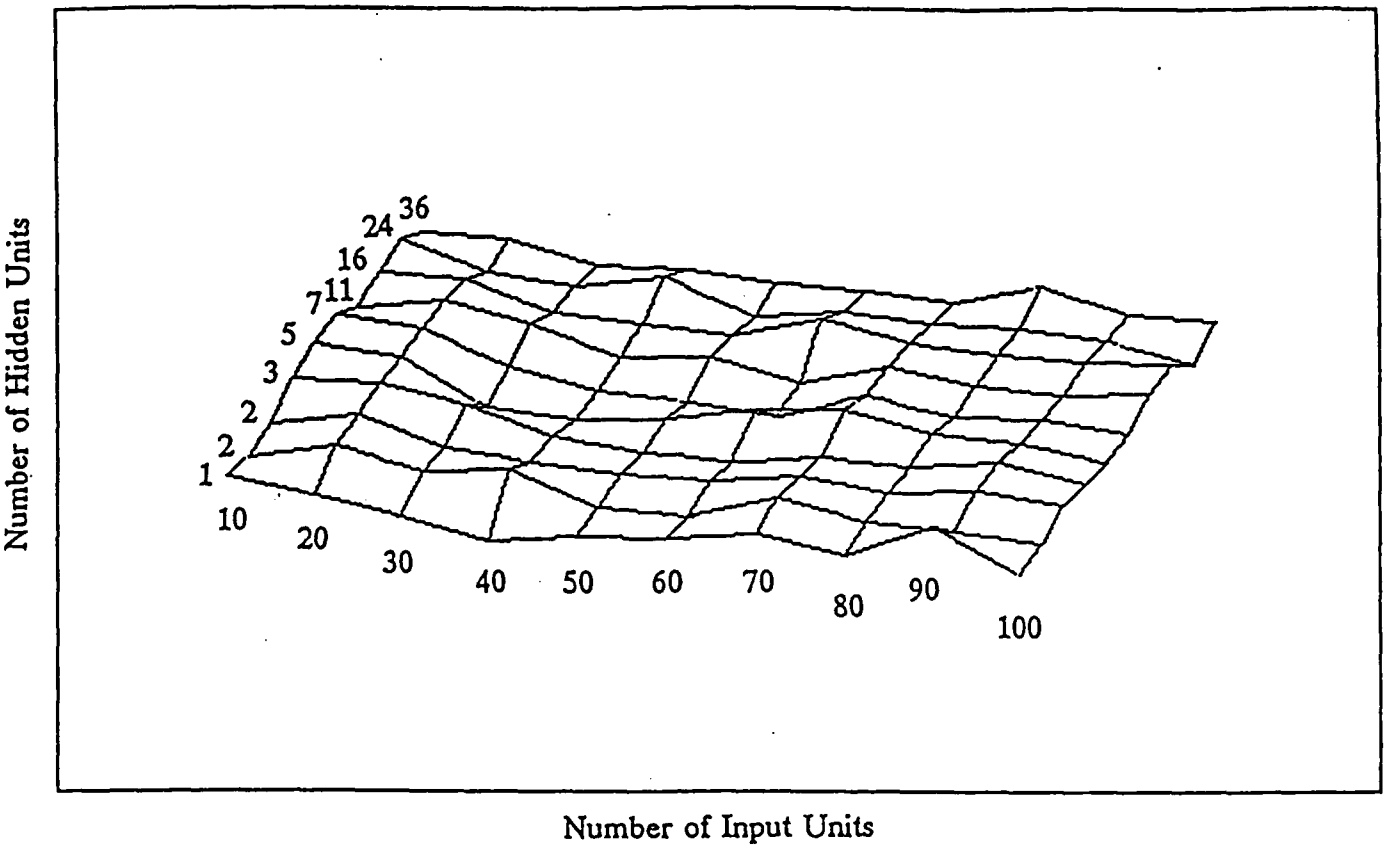

Figure 3 Percent correct highest at (10-16) = 80%.

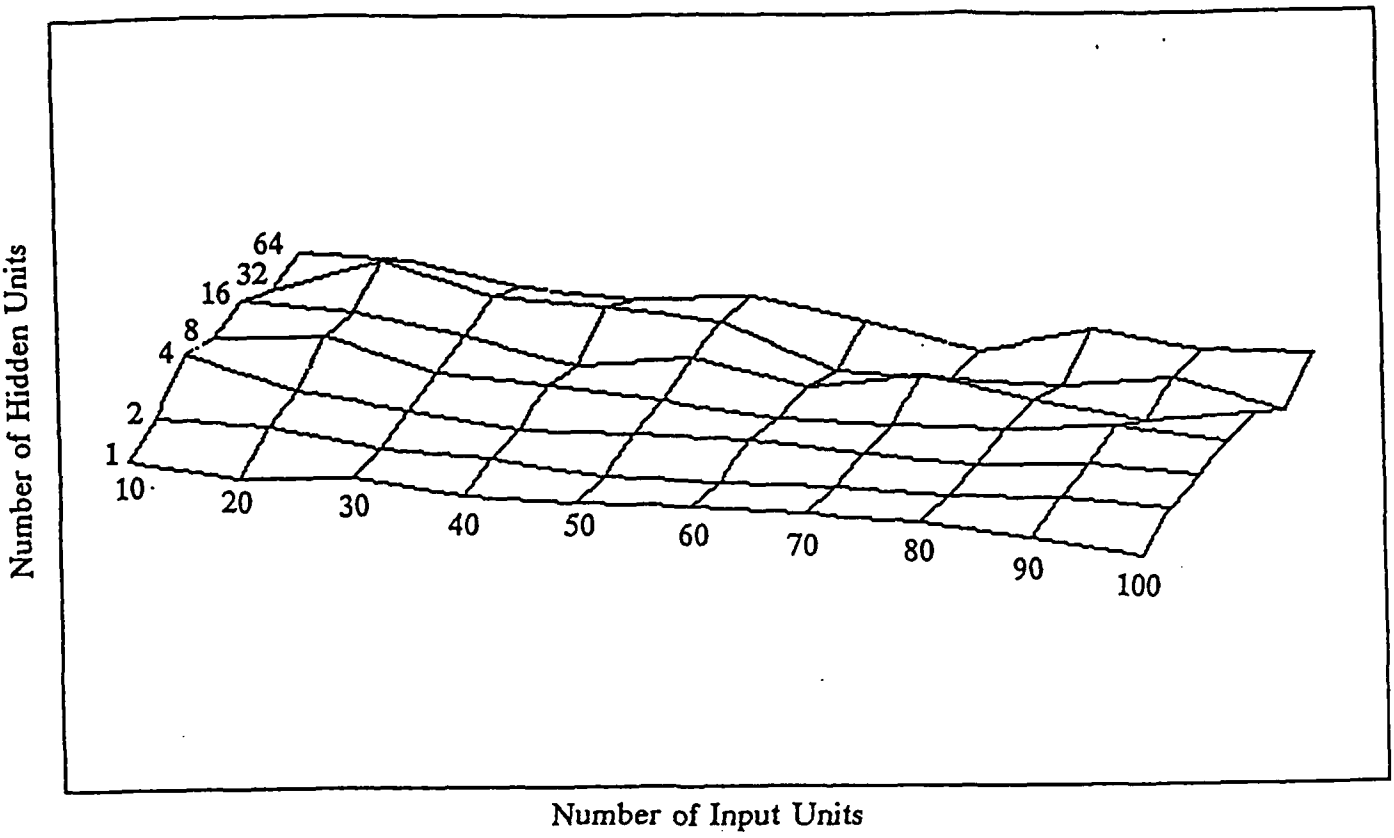

Figure 4 Percent correct highest at (20-2) = 81%.

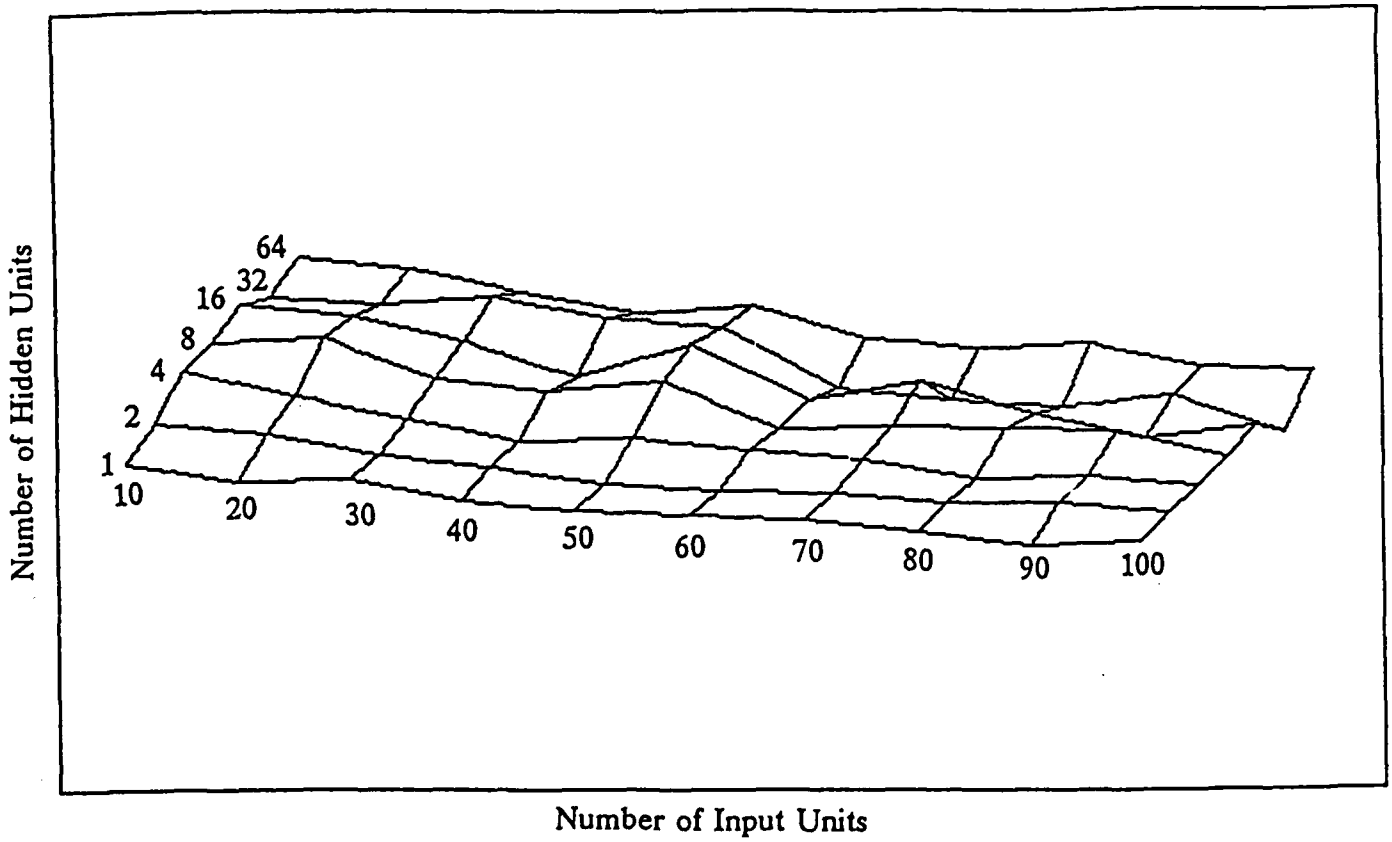

Figure 5 With +/- 10% noise, percent correct highest at (20-2) = 80%.

5.0 CAPABILITIES SUMMARY

As a result of this Phase I research program, we were able to develop a unique software package which will enable a user to easily interact with the relevant neural network technologies.

We observed that it is not necessary to implement a natural language front end to make the TUSA Workstation flexible and powerful. More importantly, the user-workstation dialogue is given a consistent context by the user interface. The structure of the interface requires each operation to be done in a specific way. Therefore, when a user interacts with the system, TUSA can predict his intentions and will then provide assistance in an intelligent manner. Among some the activities which TUSA can be expected to execute in an independent manner are:

Judging a user's epistemic state.

Suggesting alternative goal-directed strategies.

Conveying expertise in neural modeling and transient acoustics.

Along with its intelligence, our software's other strength is its flexibility. The programming language selected, SMALLTALK, is the modern standard for flexible and powerful languages. Like SMALLTALK, TUSA can adapt or can be modified by the user at many levels. Neural network paradigms can be added or existing paradigms can be modified to suit the user's needs. At the completion of Phase I, one paradigm, backpropagation, was implemented and the ease with which components of the system work and communicate together was observed firsthand.

6.0 RECOMMENDATIONS

In Phase I, we designed a framework which incorporates the fundamental elements of an intelligent tutor and we have shown how these elements interact productively. Future work will complete TUSA's knowledge about backpropagation and underwater signals. More important, however, will be the research into how diverse approaches such as expert systems, neural networks and conventional signal processing, can be controlled skillfully by a sonar operator to identify signatures faster and more accurately. A marriage of conventional signal processing methods for detection and classification with advanced neural network paradigms will surely emerge as a useful tool; iterative improvement of TUSA's design through testing with sonar operators will provide a system which will be useful in many fields (e.g. economic prediction, medical instrumentation, and robotics.)

NETROLOGIC believes that the basic feasibility of an Intelligent Sonar Analysis Tutor was demonstrated in our Phase I program. To develop an effective TUSA prototype we recommend that the following open design issues be resolved in a follow-on Phase II Study.

1) Implementation of the many backpropagation/multi-layer network rules listen in Section 2 in TUSA.

2) Expansion of the rule set to include a larger set of paradigms/architectures (e.g. ART, Hopfield Models).

3) Testing these rules with an actual sonar operator and real world sonar data.

4) Determination of the system architecture (hardware and software) best suited for real Navy applications.

5) Develop a realistic model of a sonar operator's cognitive processing so that a better interface can be produced.

7.0 REFERENCES

1. Ash, Dynamic Node Creation in Backpropagation Networks, UCSD ICS Report, 8901.

2. Baum and Haussler, What Size Net Gives Valid Generalization, Neural Computation, 1989, Vol. I, No. 1, p. 151.

3. Cater, Successfully Using Peak Learning Rates of 10 (and greater) in Backpropagation Networks with the Heuristic Learning Algorithm, ICNN Proceedings, 1987, Vol. II, p. 645.

4. Chauvin, A Backpropagation Algorithm with Optimal Use of Hidden Units, in Advances in Neural Information Processing Systems 1, Morgan Kaufmann, 1988, p. 519.

5. Cottrell, Munro and Zipser, Learning Internal Representations from Grey-scale Images, Proceedings of the 9th Annual Conference of the Cognitive Science Society, 1987.

6. Dahl, Accelerated Learning Using the Generalized Delta Rule, ICNN Proceedings, 1987, Vol. II, p. 523.

7. Durbin and Rumelhart, Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks, Neural Computation, 1989, Vol. 1, No. 1, p. 133.

8. Gutierrez, Wang and Grondin, Estimated Hidden Unit Numbers for Two Layer Perceptions, ICNN Proceedings, 1989, Vol. I, p. 677.

9. Hansen and Pratt, Comparing Biases for Minimal Network Construction with Backpropagation, in Advances in Neural Information Processing Systems 1, Morgan Kaufmann, 1988, p. 177.

10. Hush and Salas, Improving the Rate of Backpropagation with the Gradient Reuse Algorithm, ICNN Proceedings, 1988, Vol. I, p. 441.

11. Kass, R., Users Models In Dialogue Systems, eds. Kobsa, A., Wahster W., Symbolic Computation Series, Springer-Verlag, Ch. 14, 386-410, 1989

12. Kruschke, Improving Generalization in Backpropagation Networks with Distributed Bottlenecks, ICNN Proceedings, 1989, Vol. I, p. 443.

13. Kung and Wang, An Algebraic Projection Analysis for Optimal Hidden Unit Size and Learning Rate in Backpropagation Learning, ICNN Proceedings, 1988, Vol. I, p. 363.

14. McClelland and Rumelhart, Explorations in Parallel Distributed Processing, MIT Press, 1987.

15. Miikkulainen and Dyer, Forming Global Representations with Extended Backpropagation, ICNN Proceedings, 1988, Vol. 1, p. 285.

16. Mozer and Smolensky, Skeletonization: a Technique for Trimming the Fat from a Network via Relevance Assessment, Univ. Colorado - Boulder, CU-CS-421-89.

17. Owens and Filkin, Efficient Training of the Backpropagation Network by Solving a System of Stiff Ordinary Differential Equations, ICNN Proceedings, 1989, Vol II, p. 381.

18. Parker, Optimal Algorithms for Adaptive Networks: 2nd Order Backpropagation, 2nd Order Direct Propagation, 2nd Order Hebbian Learning, ICNN Proceedings, 1987, Vol. II, p. 593.

19. Pearlmutter, Learning State Space Trajectories in Recurrent Neural Networks, Neural Computation, 1989, Vol. 1, No. 2, p. 263.

20. Pineda, Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation, Neural Computation, 1989, Vol. 1, No. 2, p. 161.

21. Rumelhart and McClelland, Parallel Distributed Processing, Vol. 1 & 2, MIT Press, 1986.

22. Science Applications International, ANSim Reference Manual.

23. Sejnowski and Rosenberg, NETtalk: A Parallel Network that Learns to Read Aloud, Johns Hopkins University EECS Tech Report 86/01.

24. Self, J., Student Models in Computer Aided Instruction, Intl. J. of Man Machine Studies, No.6, 261-275, 1974.

25. Sietsma and Dow, Neural Net Pruning - Why and How, ICNN Proceedings, Vol. I, 1988, p. 325.

26. Stornetta and Huberman, Improved Three Layer Backpropagation Algorithm, ICNN Proceedings, 1987, Vol. II, p. 637.

27. Von Lehman, Paek, Liao, Marrakchi, and Patel, Factors Influencing Learning by Backpropagation, ICNN Proceedings, 1988, Vol. I, p. 335.

28. Waibel, Modular Construction of Time Delay Neural Networks for Speech Recognition, Neural Computation, 1989, Vol. 1, No. 1, p. 39.

29. Williams and Zipser, A Learning Algorithm for Continuously Running Recurrent Neural Networks, Neural Computation, 1989, Vol. 1, No. 2, p. 270.

30. Wolf, B., Building a Computer Tutor: Design Issues, Computer 17, 61-73, 1984